Fernández, Norberto; Quintás, Fernando; Sánchez, Luis; Arias, Jesús. Social noise: generating random numbers from Twitter streams. Fluctuation and Noise Letters, 2015, v.14, n.1, 1552212
https://doi.org/10.1142/S0219477515500121

Social Noise: Generating Random Numbers from Twitter Streams

Norberto Fernández, Fernando Quintas,
Luis Sánchez and Jesús Arias

Due to the multiple applications of random numbers in computer systems (cryptogra-phy, online gambling, computer simulation, etc.) it is important to have mechanisms to generate these numbers. *True Random Number Generators* (TRNGs) are commonly used for this purpose. TRNGs rely on non-deterministic sources to generate random-ness. Physical processes (like noise in semiconductors, quantum phenomenon, etc.) play this role in state of the art TRNGs. In this paper, we depart from previous work and explore the possibility of defining *social TRNGs* using the stream of public messages of the microblogging service Twitter as randomness source. Thus, we define two TRNGs based on Twitter stream information and evaluate them using the National Institute of Standards and Technology (NIST) statistical test suite. The results of the evaluation confirm the feasibility of the proposed approach.

Keywords: Random numbers; random number generator; Twitter.

1. Introduction

According to [1], a Random Number Generator (henceforth, RNG) can be defined as a *computer procedure that scrambles the bits of a current number or set of numbers to produce a new number, in such a way that the result appears to be randomly distributed among the set of possible numbers and independent of the previous generated numbers.*

RNGs have many practical applications in computer systems. For instance, they are used within crytographic protocols [2], or to generate nonces in cryptocurrencies [3]. Online games and gambling applications, like lotteries [4], rely extensively on the use of RNGs. They are also a part of simulation software [5], and are essential for Monte Carlo simulation [6] and for many graphics applications [7].

Due to its importance, the topic of random number generation has been extensively addressed in the state of the art. We can find multiple approaches for random number generation. In general, these approaches can be broadly classified into two main groups: *Pseudo Random Number Generators* (PRNGs) and *True Random Number Generators* (TRNGs). The first group includes these approaches where random variables are simulated by deterministic algorithms like, for instance, congruential random number generators [1]. The approaches in the second group use a non-deterministic source for producing the necessary randomness [8]. They usually rely on physical processes: Noise in semiconductors [9], quantum phenomenon [10], measures taken from sensors [8], etc. These two types of generators can be combined [11,12], using a TRNG to define the first value or seed of a PRNG.

In this paper, we propose a novel approach to define *social TRNGs*, exploiting a particular case of physical process: Communication between users in social networks. In particular, we explore the possibility of using the stream of public messages of the microblogging service Twitter[a] as a randomness source. Taking into account that this stream consists of an aggregate of messages published by a huge amount of users, who write about a variety of topics, use diverse languages and are located at different places, our hyphothesis is that it constitutes a non-deterministic source, adequate for our purposes.

To validate this hyphothesis, we define two TRNGs based on Twitter stream information and evaluate them using the National Institute of Standards and Technology (NIST) statistical test suite [13,14]. The results of the evaluation confirm the feasibility of the proposed approach.

The rest of this paper is organized as follows: First, in Sec. 2, we provide some background on approaches for true random number generation to contextualize our contribution. Next, in Sec. 3, we describe our TRNG approach based on information obtained from Twitter. The results of the evaluation of this approach on the NIST statistical test suite are reported in Sec. 4. Finally, Sec. 5 presents some concluding remarks and potential future lines of work.

2. Background on True Random Number Generation

As indicated in the introductory section, *True Random Number Generators* use a non-deterministic source to produce the necessary randomness. There are different approaches in the state of the art that can be classified within this group. Basically, these approaches differ on the particular randomness source used to generate

[a]http://www.twitter.com.

the numbers. In general, TRNGs rely on natural processes as randomness sources. Examples are: Noise in semiconductors [9], quantum phenomenon [10], measures taken from sensors [8], radioactive decay (used for instance by HotBits[b]), or atmospheric noise (used by Random.org[c]). However other approaches are also possible. For instance, for the purposes of this paper, the TRNGs based on information obtained from human activities are particularly relevant.

Early work addressing the generation of random numbers from human activities comes from the area of psychology. These works were mainly centered on asking a set of subjects to explicitly generate random sequences and evaluate the randomness of the results. A survey of a set of works along this line is reported in [15]. In general, the main conclusion of these experiments was that humans were unable to explicitly produce good random sequences. Further work has also explored the possibility of generating randomness indirectly from other human activities. An example along this line is, for instance, [16], where the authors explore the possibility of using competitive games as a randomness source. While the results obtained from the approach were not perfectly random, the authors were able to integrate it into a robust PRNG. The idea of extracting randomness from games is also explored in [12, 17], where the authors conclude that their true random generator approach effectively produces high quality randomness. Another example, not based on games, is the Linux random number generation system that, according to [18], relies on mouse and keyboard activity as one of its sources of randomness.

The work reported in this paper departs from previous approaches in the source of randomness to be used. The hypothesis that inspires our approach is that the information generated by the millions of users of the microblogging service Twitter can be effectively used as a source of randomness. To the best of our knowledge, this is the first approach to a TRNG that uses social network information as a noisy source of randomness.

3. Generating Random Numbers from Twitter

Though several social networking sites have proliferated in the recent past, the TRNG described in this paper uses the information provided by Twitter. There are several reasons that support our decision:

- **Popularity:** At the time of writing Twitter ranks in position 9 in the Alexa web popularity ranking.[d] Recent information provided by the company [19] indicates that it has more than 100 million daily active users.
- **Volume of data:** Twitter allows users to publish short (at most 140 characters) text messages, named *tweets*. According to [19], more than 500 million messages

[b] https://www.fourmilab.ch/hotbits/.
[c] http://www.random.org/.
[d] http://www.alexa.com/topsites.

are posted every day, and the highest peak of activity recorded at the time of writing reached more than 132,000 tweets per second.[e]

- **Availability:** The messages published by Twitter users are available to applications through the Twitter Application programming interface (API).

In particular, in this paper we use the Twitter streaming API[f] to obtain a sample of the global stream containing all the tweets publicly posted by users. Note that the tweets are messages written in human languages, which are well-known to follow certain patterns and statistical rules (like the Zipf law [20]). Thus, unappropiately chosing the procedure to extract randomness from these tweets may result in random sequences of poor quality.

Taking this into account, in this paper we explore two different procedures to generate random sequences from tweets: One is novel (*tweet length autocorrelation*), and the other is inspired by previous related work (*cryptographic hashing*):

- **Tweet length autocorrelation:** Let $T = \{t_0, t_1, \ldots\}$, represent the sequence of tweets in the stream at the input of the random generator. Let $T_E = \{t_0, t_2, \ldots\}$, represent the sequence of tweets located at an even position in T and $T_O = \{t_1, t_3, \ldots\}$, the tweets located at an odd position. Given the text of a tweet (considered as is, without any particular processing) we compute its length (in characters). Let $l(t)$ represent the function that computes the length of a certain tweet t. We define a sequence S whose elements are computed as the difference of the length of successive tweets:

$$S = \{s_0, s_1, \ldots\} = \{l(t_1) - l(t_0), l(t_3) - l(t_2), \ldots\}. \tag{1}$$

From the sequence S we obtain a new sequence S^* by simply removing the elements of S that are zero. Using S^* we obtain a sequence B of bits by applying the *sign* function to the elements of S^*:

$$sign(s_m^*) = \begin{cases} 0 & \text{if } s_m^* > 0, \\ 1 & \text{if } s_m^* < 0. \end{cases} \tag{2}$$

According to the aforementioned definitions, the elements in S can be considered as samples of a discrete random variable D that is obtained by subtracting L_O (discrete random variable that represents the length of the tweets in odd position) and L_E (discrete random variable that represents the length of the tweets in even position), that is, $D = L_O - L_E$. Assuming that L_E and L_O are independent variables (tweets are either in an even or odd position in the stream), the distribution of the difference of L_E and L_O can be computed as:

$$p_D[n] = \sum_{k=-\infty}^{+\infty} p_{L_O}[k] \cdot p_{L_E}[k-n]. \tag{3}$$

[e] https://blog.twitter.com/2013/new-tweets-per-second-record-and-how.
[f] https://dev.twitter.com/docs/streaming-apis.

If we take into account that L_E and L_O are also identically distributed variables (both follow the general distribution of the length of tweets L) we obtain:

$$p_D[n] = \sum_{k=-\infty}^{+\infty} p_L[k] \cdot p_L[k-n]. \tag{4}$$

Thus, it can be shown that, in practice, according to Eq. (4), the probability distribution of D is the autocorrelation function of L. As one of the properties of autocorrelation functions is that they are symmetric with respect to the Y-axis, the probability $P(D[k] > 0)$ is equal to the probability $P(D[k] < 0)$. Due to this, it can be expected that S would consists of a balanced mixture of strictly positive ($s_i > 0$) and strictly negative ($s_i < 0$) samples, and the same can be expected for S^* (because S^* is obtained from S by removing the samples that are zero). Due to this property, the number of 0 and 1 s of the binary sequence B should be balanced, a necessary (though not sufficient) condition to have a good random generator.

Figure 1 shows the distribution of L_E (left), a Q-Q plot comparing the distributions of L_E and L_O (center), and the distribution of $D = L_O - L_E$ (right) obtained from a dataset of 2 million tweets gathered from the Twitter streaming API. As expected, we obtain a line $Y = X$ for the Q-Q plot, and a symmetric distribution around zero for D. Note that the distribution of L_E is not uniform, with a clear maximum when the tweet length is equal to 140 characters. This is the maximum tweet length allowed by Twitter and, thus, many software applications truncate longer messages to this length before publishing them in the micro-blogging service.

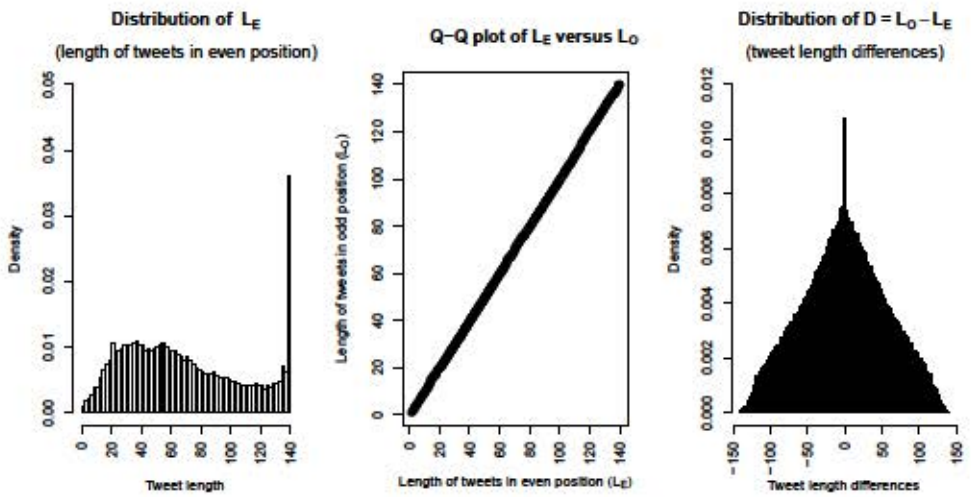

Fig. 1. Distribution of L_E (left), Q-Q plot of L_E versus L_O (center) and distribution of D (right).

- **Cryptographic hashing:** The use of cryptographic hash functions as part of random number generators has been suggested by previous work like, for instance, [7, 8]. In this paper, we explore the viability of using these functions to generate random sequences from tweets. To do so, we concatenate the UTF-8 text of a sequence of tweets and process that text using the SHA-3[g] cryptographic hash function. In particular, we apply the SHA-3 function to a buffer of 1024 bytes of text to obtain a 64 byte hash value.

4. Evaluation

As indicated in [14], randomness is a probabilistic property and, thus, statistical tests can be used to evaluate the quality of a random sequence. Following previous work in the state of the art, like [8], we rely on the statistical test suite defined by the NIST [14] to evaluate the quality of the random numbers generated with our approach. The NIST suite defines 15 tests, each of them using a different approach to

Table 1. Description of the tests included within the NIST suite (according to [13, 14]).

Test	Purpose
Frequency (Monobit)	Unbalanced number of zeros and ones.
Frequency within a block	Unbalanced number of zeros and ones within blocks of certain size M.
Runs	The oscillation between zeros and ones is too fast or slow.
Longest run of ones in a block	Deviation of the distribution of long runs of ones from the expected value for random sequences.
Binary matrix rank	Linear dependence among fixed length bit subsequences.
Spectral	Periodic features (i.e. repetitive patterns)
Non-overlapping template matching	There are too many occurrences of aperiodic patterns of size m.
Overlapping template matching	There are too many occurrences of runs of ones of size m.
Maurer's "Universal Statistical"	A random sequence should not be significantly compressed without loss of information.
Linear complexity	Compute complexity using the Berlekamp–Massey [21] algorithm. Random sequences should be complex enough.
Serial	Deviation from the number of occurrences of overlapping patterns of m bits expected for random sequences.
Approximate entropy	Deviation from the number of occurrences of overlapping patterns of consecutive lengths (m, $m+1$) expected for random sequences.
Cumulative sums	Replace by $(-1, +1)$ the bits in the sequence and compute the cumulative sums of subsequences of bits. The result for random sequences should be near zero.
Random excursions	Deviation from the distribution of the number of visits of a random walk to a certain state.
Random excursions variant	Deviation from the distribution of the total number of visits across many random walks to a certain state.

[g]http://csrc.nist.gov/groups/ST/hash/sha-3/sha-3_standardization.html.

detect cases of non-randomness in the binary sequences under evaluation. Table 1 lists the names of the different tests and indicates their purpose (cause of non-randomness that they are designed to detect), as reported in [13,14].

The NIST randomness evaluation suite is based on hypothesis testing. In particular, the *null hypothesis* to be validated (or rejected) is that the binary sequence being processed is random. When a test is run on a binary sequence, a test statistic value (p-value) is computed. This p-value is compared with a predefined *significance level* (α) to decide whether the sequence has passed the test or not.

Obviously, in order to evaluate the quality of a random number generator we have to test several sequences extracted from that generator. Thus, several p-values are obtained and should be checked. In particular, according to NIST recommendations, the number of sequences to be tested should be in the order of the inverse of the significance level (that is, α^{-1}).

In order to interpret the results obtained by running a certain test on a set of binary sequences, the NIST recommends two different approaches [14]:

- First, the proportion of binary sequences that pass the test (that we denote as P) should be computed:

$$P = \frac{\text{Sequences that pass the test}}{\text{Total number of sequences being tested}}. \tag{5}$$

  According to [14] the test is successfully passed if this proportion is included in the interval:

$$P \in \left[\hat{p} - 3\sqrt{\frac{\hat{p}(1-\hat{p})}{N}}, \hat{p} + 3\sqrt{\frac{\hat{p}(1-\hat{p})}{N}}\right], \tag{6}$$

  where $\hat{p} = 1 - \alpha$ and N is the total number of sequences being tested.
- Second, the distribution of the p-values obtained for the different sequences should be examined in order to ensure its uniformity. A Chi-square (χ^2) goodness-of-fit distributional test is used to accomplish this goal. In particular, the interval between 0 and 1 is divided into 10 sub-intervals and the number of p-values lying in each sub-interval is computed. Then, the χ^2 value is computed as:

$$\chi^2 = \sum_{i=1}^{10} \frac{(F_i - (N/10))^2}{N/10}, \tag{7}$$

  where F_i represents the number of p-values in each interval i and N is the number of sequences being tested. The p-value for this χ^2 test can be computed as:

$$p\text{-value}_\chi = igamc(9/2, \chi^2/2), \tag{8}$$

  where *igamc* represents the incomplete gamma function:

$$igamc(a,x) = \frac{1}{\Gamma(a)} \int_x^\infty e^{-t} t^{a-1} dt \quad \text{with } \Gamma(a) = \int_0^\infty e^{-t} t^{a-1} dt. \tag{9}$$

  If $p\text{-value}_\chi \geq 0.0001$ the distribution of p-values can be considered to be uniform and the test is successfully passed.

If any of the two aforementioned approaches fails to produce conclusive results, further sequences need to be obtained from the random generator and tested, to discriminate between actual non-randomness and potential statistical anomalies.

In order to generate the binary sequences to be evaluated using the NIST suite, we followed a two-step process: (1) We used the Twitter streaming API to capture the tweets in the global stream from 26 February 2014 to 26 May 2014; (2) using this dataset, we followed the mechanisms presented in Sec. 3 to generate the binary sequences to be tested. In particular, for each of the methods described in Sec. 3 we generated 100 binary sequences[h] (i.e., $N = 100$) of 1 million bits (values suggested by the NIST in [14] and used also in [8]).

Once the random sequences were generated, we analyzed them with the NIST test suite. The suite provides as output a log file[i] reporting the values of P and $p\text{-value}_\chi$ for each test. An important aspect to note is that the NIST software automatically repeats some tests (for instance, the *Non-overlapping template* or the *Random excursions variant*) several times with different configurations. Due to this, these tests have several values for P and $p\text{-value}_\chi$. Another aspect to take into account is that for the *Random excursions* and *Random excursions variant* tests the NIST software uses only a subset of the available sequences (in particular, in these two cases, $N = 62$).

Taking into account that the significance level was set to $\alpha = 0.01$ (as suggested in [8,14]), in order to consider a particular test as passed it should happen that:

- In order to satisfy the restrictions imposed by Eq. (6), when $N = 100$ it should happen that $P \in [0.960, 1.020]$, whereas when $N = 62$ it should happen that $P \in [0.952, 1.028]$. As P cannot be greater than 1, it is enough to check that $P \geq 0.960$ when $N = 100$ and that $P \geq 0.952$ when $N = 62$.
- $p\text{-value}_\chi \geq 0.0001$.

Table 2 summarizes the results (P and $p\text{-value}_\chi$) reported in the log files of the NIST suite. When a test has several values for P and $p\text{-value}_\chi$, we just report in the table the worst case (the minimum values obtained). According to the results in Table 2, it can be seen that both the *Tweet length autocorrelation* and *Cryptographic hashing* approaches successfully pass all the tests.

Another aspect to be evaluated is the rate at which the random information is generated by the two approaches. Table 3 shows the average binary rate (random bits at the output produced per bit of input information) and average per-tweet rate (random bits at the output per tweet at the input) measured for each of the approaches. Taking into account that the average tweet rate reported by Twitter[j] in 2013 was around 5,700 tweets per second, we would obtain an effective random bit rate of 2.7 Kbps when using the tweet length autocorrelation approach, and a

[h]The sequences are available at: http://www.it.uc3m.es/berto/SocialNoise/.
[i]The log files are available at: http://www.it.uc3m.es/berto/SocialNoise/.
[j]https://blog.twitter.com/2013/new-tweets-per-second-record-and-how.

Table 2. Results obtained in the empirical evaluation with NIST test suite.

Test	Tweet length autocorrelation		Cryptographic hashing	
	P	p-value$_\chi$	P	p-value$_\chi$
Frequency (Monobit)	0.980	0.262	0.980	0.097
Frequency within a block	0.990	0.350	0.990	0.534
Runs	0.990	0.015	0.990	0.720
Longest run of ones	1	0.419	1	0.798
Binary matrix rank	1	0.798	0.990	0.534
Spectral	0.980	0.475	0.980	0.154
Non-overlapping template	0.960	0.001	0.960	0.027
Overlapping template matching	0.980	0.679	0.960	0.419
Maurer's "Universal Statistical"	0.980	0.494	0.960	0.097
Linear complexity	0.980	0.972	1	0.367
Serial	0.990	0.091	0.990	0.276
Approximate entropy	1	0.033	0.970	0.798
Cumulative sums	0.970	0.911	0.980	0.154
Random excursions	0.952	0.091	0.968	0.016
Random excursions variant	0.968	0.035	0.968	0.028

Table 3. Comparison of approaches regarding the rate of generation of random information.

Approach	Average binary rate	Average per-tweet rate
Tweet Length Autocorrelation	0.000675	0.489
Cryptographic hashing	0.0625	45.329

rate of 252.3 Kbps in the case of cryptographic hashing. As a reference, the HotBits developers report a rate of around 800 bits per second[k] for their TRNG. Note that, in case an application requires higher random bit rates, it is possible to combine the TRNG approaches in this paper with a PRNG [11,12], using the non-deterministic random numbers they generate to periodically seed the PRNG.

Analyzing the results reported in Table 3, it can be seen that the per-tweet rate of the tweet length autocorrelation approach is slightly less than the theoretical 0.5. The reason is that we do not generate any bit when the difference between successive tweet lengths is zero and, thus, in practice some tweets are not used to generate random bits. Note also that, as expected, the approach based on cryptographic hashing produces a higher bit rate, because it takes advantage of the contents of the tweets, not only their length.

5. Conclusions and Perspectives

In this paper, we proposed a novel approach to define *social TRNGs*: A particular case of TRNGs based on the use of social network information (the stream of

[k]https://www.fourmilab.ch/hotbits/.

public messages in Twitter) as randomness source. We defined two TRNGs based on Twitter stream information and evaluated them using the NIST statistical test suite, with positive results.

Though the empirical evaluation indicates that the binary sequences generated by our TRNGs are random, we have to take into account that we use a public information source (Twitter) to generate them. Obviously, this introduces some caveats with respect to its potential use in application scenarios where security is an important issue. Thus, conducting a detailed analysis of the security implications of using random numbers generated with our approach is an interesting future line of work to consider.

Acknowledgement

This work has been partially funded by the Spanish Ministerio de Economa y Competitividad through the project HERMES-SMARTDRIVER (TIN2013-46801-C4-2-R).

References

[1] G. Marsaglia, Random number generation, in *Encyclopedia of Computer Science* (John Wiley and Sons Ltd., 2003), pp. 1499–1503.

[2] A. Freier and P. Karlton, Request for comments 6101, The secure sockets layer (SSL) Protocol Version 3.0, 2011.

[3] D. Hobson, What is bitcoin? *XRDS* **20** (2013) 40–44.

[4] S. S. M. Chow, L. C. K. Hui, S. M. Yiu and K. P. Chow, An e-lottery scheme using verifiable random function, in *Proc. of the 2005 Int. Conf. on Computational Science and Its Applications — Volume Part III* (2005), ICCSA'05, pp. 651–660.

[5] P. L'Ecuyer, Random numbers for simulation, *Commun. ACM* **33** (1990) 85–97.

[6] I. Lee, Parallel random number generations for Monte Carlo simulation, in *Proc. of the 49th Annual Southeast Regional Conf.* (2011), ACM-SE '11, pp. 330–331.

[7] S. Tzeng and L.-Y. Wei, Parallel white noise generation on a GPU via cryptographic hash, in *Proc. of the 2008 Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2008), I3D '08, pp. 79–87.

[8] V. Gaglio, A. De Paola, M. Ortolani and G. Lo Re, A TRNG exploiting multi-source physical data, in *Proc. of the 6th ACM Workshop on QoS and Security for Wireless and Mobile Networks* (2010), Q2SWinet '10, pp. 82–89.

[9] J. Hong, S. Park, J. Yoon, J. Koh and D. Kim, Design of random noise generator using SW algorithm, in *Proc. of the 1st Int. Symp. Information and Communication Technologies* (2003), ISICT '03, pp. 9–14.

[10] H. A. Shenoy, R. Srikanth and T. Srinivas, Efficient quantum random number generation using quantum indistinguishability, *Fluct. Noise Lett.* **12** (2013) 1350020.

[11] P. L'Ecuyer, Random number generation, in *Handbook of Computational Statistics*, Springer Handbooks of Computational Statistics (Springer, 2012), pp. 35–71.

[12] M. Alimomeni and R. Safavi-Naini, Human assisted randomness generation using video games, *IACR Cryptology ePrint Archive* **2014** (2014) 45.

[13] J. Soto, Statistical testing of random number generators, in *Proc. of the 22nd National Information Systems Security Conf.* (1999).

[14] National Institute of Standards and Technology (NIST), A statistical test suite for the validation of random number generators and pseudo random number generators for cryptographic applications (2010).
[15] W. A. Wagenaar, Generation of random sequences by human subjects: A critical survey of literature, *Psychol. Bull.* **77** (1972) 65–72.
[16] R. Halprin and M. Naor, Games for extracting randomness, in *Proc. of the 5th Symp. Usable Privacy and Security* (2009), SOUPS '09, pp. 12:1–12:12.
[17] M. Alimomeni, R. Safavi-Naini and S. Sharifian, A true random generator using human gameplay, in *Decision and Game Theory for Security*, Lecture Notes in Computer Science, Vol. 8252 (Springer International Publishing, 2013), pp. 10–28.
[18] Z. Gutterman, B. Pinkas and T. Reinman, Analysis of the linux random number generator, in *Proc. of the 2006 IEEE Symp. on Security and Privacy* (2006), SP '06, pp. 371–385.
[19] Twitter Inc., United states securities and exchange commission (SEC) amendment no. 4 to form s-1 (2013).
[20] G. K. Zipf, *The Psychobiology of Language* (Houghton-Mifflin, 1935).
[21] A. J. Menezes, S. A. Vanstone and P. C. V. Oorschot, *Handbook of Applied Cryptography*, 1st edn. (CRC Press, Inc., 1996).